

IDENTITIES IN PLACTIC, HYPOPLACTIC, SYLVESTER, BAXTER, AND RELATED MONOIDS

ALAN J. CAIN AND ANTÓNIO MALHEIRO

ABSTRACT. This paper considers whether non-trivial identities are satisfied by certain ‘plactic-like’ monoids that, like the plactic monoid, are closely connected with combinatorics. New results show that the hypoplactic, sylvester, Baxter, stalactic, and taiga monoids satisfy identities. The existing state of knowledge is discussed for the plactic and Bell monoids.

1. INTRODUCTION

The ubiquitous plactic monoid, whose elements can be viewed as semi-standard Young tableaux, and which appears in such diverse contexts as symmetric functions [Mac08], representation theory [Ful97], algebraic combinatorics [Lot02], Kostka–Foulkes polynomials [LS81, LS78], Schubert polynomials [LS85, LS90], and musical theory [Jed11], is one of a family of ‘plactic-like’ monoids that are closely connected with combinatorics. These monoids include the hypoplactic [Nov00], sylvester [HNT05], taiga [Pri13, § 5], stalactic [Pri13], Baxter [Gir12], and Bell [Rey07] monoids. Each of these monoids is obtained by factoring the free monoid \mathcal{A}^* over the infinite ordered alphabet $\mathcal{A} = \{1 < 2 < 3 < \dots\}$ by a congruence that arises from a so-called insertion algorithm that computes a combinatorial object from a word. For instance, for the plactic monoid, the corresponding combinatorial objects are (semi)standard Young tableaux; for the sylvester monoid, they are binary search trees.

An *identity* is a formal equality between two words in the free monoid, and is *non-trivial* if the two words are distinct. A monoid M *satisfies* such an identity if the equality in M holds under every substitution of letters in the words by elements of M . For example, any commutative monoid satisfies the non-trivial identity $xy = yx$.

Jaszuńska & Okniński [JO11, Corollary 3.3.4] proved that the related Chinese monoid [CEK⁺01], which has the same growth type as the plactic monoid [DK94] but which does not arise from such a natural combinatorial object, satisfies Adian’s identity $xyxyxyxyx = xyxyxyxyx$ (which is the shortest non-trivial identity satisfied by the bicyclic monoid [Adi66, Chapter IV, Theorem 2]). This naturally leads to the question of whether the plactic

The first author was supported by an Investigador FCT fellowship (IF/01622/2013/CP1161/CT0001). For both authors, this work was partially supported by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the project UID/MAT/00297/2013 (Centro de Matemática e Aplicações), and the project PTDC/MHC-FIL/2583/2014.

TABLE 1. Examples of non-trivial identities satisfied by ‘plactic-like’ monoids.

<i>Monoid</i>	<i>Symbol</i>	<i>Identity satisfied</i>	<i>Discussed in</i>	<i>Citation</i>
Plactic	plac	?	Subsec. 3.7	—
Hypoplactic	hypo	$xyxy = yxyx$	Subsec. 3.1	Present paper
Sylvester	sylv	$xyxy = yxxy$	Subsec. 3.3	Present paper
Baxter	baxt	$xyxyxy = yxyxxy$	Subsec. 3.5	Present paper
Stalactic	stal	$xyxy = yxxy$	Subsec. 3.4	Present paper
Taiga	taig	$xyxy = yxxy$	Subsec. 3.2	Present paper
Bell	bell	None	Subsec. 3.6	[CMS]

monoid, and the other monoids in the family discussed above, satisfy non-trivial identities.

The goal of this paper is to present new results showing that some of these monoids satisfy non-trivial identities, and to survey the state of knowledge for other monoids in this family. New results show that the hypoplactic, sylvester, baxter, stalactic, and taiga monoids satisfy non-trivial identities. A discussion of the situation for the Bell and plactic monoids completes the paper. Table 1 summarizes the results.

2. ‘PLACTIC-LIKE’ MONOIDS

In this section, we recall only the definition and essential facts about the various monoids; for further background, see [Lot02, Ch. 5] on the plactic monoid, [Nov00] on the hypoplactic monoid, [HNT05] on the sylvester monoid; [Pri13, § 5] on the taiga monoid; [Pri13] on the stalactic monoid; [Gir12] on the Baxter monoid; and [Rey07] on the Bell monoid.

2.1. Alphabets and words. For any alphabet X , the free monoid (that is, the set of all words, including the empty word) on the alphabet X is denoted X^* . The empty word is denoted ε . For any $u \in X^*$, the length of u is denoted $|u|$, and, for any $x \in X$, the number of times the symbol x appears in u is denoted $|u|_x$.

Throughout the paper, $\mathcal{A} = \{1 < 2 < 3 < \dots\}$ is the set of natural numbers viewed as an infinite ordered alphabet, and $\mathcal{A}_n = \{1 < 2 < \dots < n\}$ is set of the first n natural numbers viewed as a finite ordered alphabet.

2.2. Combinatorial objects and insertion algorithms. A *Young tableau* is a finite array of symbols from \mathcal{A} , with rows non-decreasing from left to right and columns strictly increasing from top to bottom, with shorter rows below longer ones, and with rows left-justified. An example of a Young tableau is

$$(2.1) \quad \begin{array}{|c|c|c|c|} \hline 1 & 1 & 2 & 3 \\ \hline 2 & 2 & 3 & \\ \hline 5 & 6 & & \\ \hline \end{array}.$$

The following algorithm takes a Young tableau and a symbol from \mathcal{A} and yields a new Young tableau:

Algorithm 2.1 (Schensted’s algorithm). *Input:* A Young tableau T and a symbol $a \in A$.

- (1) If a is greater than or equal to every entry in the topmost row of T , add a as an entry at the rightmost end of T and output the resulting tableau.
- (2) Otherwise, let z be the leftmost entry in the top row of T that is strictly greater than a . Replace z by a in the topmost row and recursively insert z into the tableau formed by the rows of T below the topmost. (Note that the recursion may end with an insertion into an ‘empty row’ below the existing rows of T .)

A *quasi-ribbon tableau* is a finite array of symbols from \mathcal{A} , with rows non-decreasing from left to right and columns strictly increasing from top to bottom, that does not contain any 2×2 subarray (that is, of the form $\begin{smallmatrix} \Box & \Box \\ \Box & \Box \end{smallmatrix}$). An example of a quasi-ribbon tableau is:

$$(2.2) \quad \begin{array}{|c|c|c|} \hline 1 & 1 & 2 \\ \hline & 3 & 4 & 4 \\ \hline & & 5 \\ \hline & & 6 & 6 \\ \hline \end{array} .$$

Notice that the same symbol cannot appear in two different rows of a quasi-ribbon tableau. There is also an insertion algorithm for quasi-ribbon tableau:

Algorithm 2.2 ([Nov00, Algorithm 4.4]). *Input:* A quasi-ribbon tableau T and a symbol $a \in \mathcal{A}$.

If there is no entry in T that is less than or equal to a , output the tableau obtained by putting a and gluing T by its top-leftmost entry to the bottom of a .

Otherwise, let x be the right-most and bottom-most entry of T that is less than or equal to x . Put a new entry a to the right of x and glue the remaining part of T (below and to the right of x) onto the bottom of the new entry a . Output the new tableau.

A (*right strict*) *binary search tree* is a labelled rooted binary tree where the label of each node is greater than or equal to the label of every node in its left subtree, and strictly less than every node in its right subtree. An example of a binary search tree is

$$(2.3) \quad \begin{array}{c} (4) \\ \swarrow \quad \searrow \\ (2) \quad (5) \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ (1) \quad (4) \quad (5) \quad (6) \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ (1) \quad (3) \quad (5) \quad (7) \end{array} .$$

The insertion algorithm for binary search trees adds the new symbol as a leaf node in the unique place that maintains the property of being a binary search tree:

Algorithm 2.3 ([HNT05, § 3.3]). *Input:* A binary search tree T and a symbol $a \in \mathcal{A}$.

If T is empty, create a node and label it a . If T is non-empty, examine the label x of the root node; if $a \leq x$, recursively insert a into the left subtree

of the root node; otherwise recursively insert a into the right subtree of the root node. Output the resulting tree.

A *stalactic tableau* is a finite array of symbols of \mathcal{A} in which columns are top-aligned, and two symbols appear in the same column if and only if they are equal. For example,

$$(2.4) \quad \begin{array}{|c|c|c|c|c|} \hline 3 & 1 & 2 & 6 & 5 \\ \hline 3 & 1 & & 6 & 5 \\ \hline & 1 & & & \\ \hline \end{array}$$

is a stalactic tableau. The insertion algorithm is very straightforward:

Algorithm 2.4 ([HNT07, § 3.7]). *Input:* A stalactic tableau T and a symbol $a \in \mathcal{A}$.

If a does not appear in T , add a to the left of the top row of T . If a does appear in T , add a to the bottom of the (by definition, unique) column in which a appears. Output the new tableau.

A *binary search tree with multiplicities* is a labelled binary search tree in which each label appears at most once, and where a non-negative integer called the *multiplicity* is assigned to each node label. An example of a binary search tree is:

$$(2.5) \quad \begin{array}{c} (4^1) \\ \swarrow \quad \searrow \\ (2^1) \quad (5^3) \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ (1^2) \quad (3^1) \quad (6^2) \end{array} .$$

(The superscripts on the labels in each node denote the multiplicities.)

Algorithm 2.5. ([Pri13, Algorithm 3]) *Input:* A binary search tree with multiplicities T and a symbol $a \in \mathcal{A}$.

If T is empty, create a node, label it by a , and assign it multiplicity 1. If T is non-empty, examine the label x of the root node; if $a < x$, recursively insert a into the left subtree of the root node; if $a > x$, recursively insert a into the right subtree of the root node; if $a = x$, increment by 1 the multiplicity of the node label x .

A *left strict binary search tree* is a labelled rooted binary tree where the label of each node is strictly greater than the label of every node in its left subtree, and less than or equal to every node in its right subtree; see the left tree shown in (2.6) below for an example.

The *canopy* of a (right or left strict) binary search tree T is the word over $\{0, 1\}$ obtained by traversing the empty subtrees of the nodes of T from left to right, except the first and the last, labelling an empty left subtree by 1 and an empty right subtree by 0. (See (2.6) below for examples of canopies.)

A *pair of twin binary search trees* consist of a left strict binary search tree T_L and a right strict binary search tree T_R , such that T_L and T_R contain the same symbols, and the canopies of T_L and T_R are complementary, in the sense that the i -th symbol of the canopy of T_L is 0 (respectively 1) if and only if the i -th symbol of the canopy of T_R is 1 (respectively 0). The following is an example of a pair of twin binary search trees, with the complementary

canopies 0110101 and 1001010 shown in grey:

$$(2.6) \quad \left(\begin{array}{c} \text{Tree 1} \quad \text{Tree 2} \end{array} \right) .$$

Tree 1: Root 4. Left child 2, right child 5. Node 2 has left child 1 (0) and right child 3 (1). Node 3 has right child 3 (1, 0). Node 5 has left child 4 (1, 0) and right child 6 (1).
Tree 2: Root 3. Left child 1, right child 4. Node 1 has right child 3 (0). Node 3 has left child 2 (1, 0). Node 4 has left child 4 (1, 0) and right child 6 (1, 0). Node 6 has left child 5 (1, 0).

A *Bell tableau* is a finite array of symbols of \mathcal{A} in which columns are top-aligned, the entries in the top row are non-decreasing from left to right, and the entries in each column are strictly increasing from top to bottom. For example,

$$(2.7) \quad \begin{array}{|c|c|c|c|} \hline 1 & 1 & 2 & 4 \\ \hline 3 & & 3 & 5 \\ \hline & & 7 & 6 \\ \hline & & 8 & \\ \hline \end{array}$$

is a Bell tableau. The insertion algorithm is very straightforward:

Algorithm 2.6 ([Rey07, Algorithm 2.6]). *Input:* A Bell tableau T and a symbol $a \in \mathcal{A}$.

If a is greater than every symbol that appears in the top row of T , add a to the right of the top row of T . Otherwise, let C be the leftmost column whose topmost symbol is strictly greater than a . Slide column C down by one space and add a as a new entry on top of C . Output the new tableau.

2.3. Monoids from insertion. Let $X \in \{\text{plac}, \text{hypo}, \text{sylv}, \text{taig}, \text{stal}, \text{baxt}, \text{bell}\}$ and $u \in \mathcal{A}^*$. Using the insertion algorithms described above, one can compute from u a combinatorial object $P_X(u)$ of the type associated to X in Table 2. If $X \neq \text{baxt}$, one computes $P_X(u)$ by starting with the empty combinatorial object and inserting the symbols of u one-by-one using the appropriate insertion algorithm and proceeding through the word u either left-to-right or right-to-left as shown in the table. For $X = \text{baxt}$, one uses a slightly different procedure: $P_{\text{baxt}}(u)$ is the pair of twin binary search trees (T_L, T_R) , where T_L is the left strict binary search tree obtained by starting with the empty tree and inserting the symbols of u one-by-one using the natural analogue of Algorithm 2.3, proceeding *left-to-right* through u , and where T_R is $P_{\text{sylv}}(u)$.

For each $X \in \{\text{plac}, \text{hypo}, \text{sylv}, \text{taig}, \text{stal}, \text{baxt}, \text{bell}\}$, define the relation \equiv_X by

$$u \equiv_X v \iff P_X(u) = P_X(v).$$

In each case, the relation \equiv_X is a congruence on \mathcal{A}^* , and so the factor monoid \mathcal{A}^*/\equiv_X can be formed, and is named as in Table 2.

It follows from the definition of \equiv_X (for any $X \in \{\text{plac}, \text{hypo}, \text{sylv}, \text{taig}, \text{stal}, \text{baxt}, \text{bell}\}$) that each element $[u]_{\equiv_X}$ of the factor monoid X can be identified with the combinatorial object $P_X(u)$.

The *evaluation* (also called the *content*) of a word $u \in \mathcal{A}^*$, denoted $\text{ev}(u)$, is the infinite tuple of non-negative integers, indexed by \mathcal{A} , whose a -th element is $|u|_a$; thus this tuple describes the number of each symbol in \mathcal{A}

<i>Monoid</i>	<i>Sym.</i>	<i>Combinatorial object</i>	<i>Alg.</i>	<i>Direc.</i>	<i>Example</i>
Plactic	plac	Young tableau	2.1	L-to-R	$P_{\text{plac}}(252163123)$ is (2.1)
Hypoplactic	hypo	Quasi-ribbon tableau	2.2	L-to-R	$P_{\text{hypo}}(615314624)$ is (2.2)
Sylvester	sylv	Binary search tree	2.3	R-to-L	$P_{\text{sylv}}(3541712654)$ is (2.3)
Stalactic	stal	Stalactic tableau	2.4	R-to-L	$P_{\text{stal}}(3613112565)$ is (2.4)
Taiga	taig	BST with mult.	2.5	R-to-L	$P_{\text{taig}}(251653254)$ is (2.5)
Baxter	baxt	Pair of twin BSTs	—	—	$P_{\text{baxt}}(42531643)$ is (2.6)
Bell	bell	Bell tableau	2.6	L-to-R	$P_{\text{bell}}(311873542)$ is (2.7)

TABLE 2. Insertion algorithms used to compute combinatorial objects, and the corresponding monoids. ‘L-to-R’ and ‘R-to-L’ abbreviate ‘left-to-right’ and ‘right-to-left’, respectively. The algorithm used to compute $P_{\text{baxt}}(\cdot)$ is a special case and is discussed in the main text.

that appears in u . It is immediate from the definition of the monoids above that if $u \equiv_{\mathbf{x}} v$, then $\text{ev}(u) = \text{ev}(v)$, and hence it makes sense to define the evaluation of an element p of one of these monoids to be the evaluation of any word representing it.

3. IDENTITIES

Subsections 3.1 to 3.5 prove that non-trivial identities are satisfied by **hypo**, **taig**, **sylv**, **baxt**, and **stal**, in that order. There are surjective homomorphisms from **baxt** onto **sylv**; from **sylv** onto **taig**, and from **stal** onto **taig**. If a monoid satisfies a particular identity, then any of its homomorphic images are also satisfy that identity. Thus it would be possible to deduce all the results from the results for **baxt** and **stal**. However, a direct proof for **baxt** would essentially subsume the proof for **sylv**, so it seems better to prove this result separately, and the simplicity of the direct proof for **taig** contrasts with the more complex proof for **sylv**, despite elements of both monoids being trees. Thus direct proofs of each result are given. For **taig**, **sylv**, **baxt**, and **stal**, the same general strategy is followed in each case: first, it is shown that an ‘identity’ holds where substitutions are limited to elements that have the same evaluation; then, by replacing each letter of the ‘identity’ by xy or yx , a non-trivial identity in the usual sense is obtained.

Subsection 3.6 summarizes the reason that **bell** does not satisfy a non-trivial identity, and Subsection 3.7 discuss the current state of knowledge for **plac**.

3.1. Hypoplactic monoid. The authors proved the following result using a quasi-crystal structure for the hypoplactic monoid [CM, Theorem 9.3]; this subsection presents a direct proof.

Proposition 3.1. *The hypoplactic monoid satisfies the non-trivial identities $xyxy = yxxy = yxyx = xyxy$.*

Proof. Let $x, y \in \mathbf{hypo}$, and let $u, v \in \mathcal{A}^*$ be words representing x, y , respectively. Let $B = \{a_1 < \dots < a_k\}$ be the set of symbols in \mathcal{A} that appear in at least one of u and v .

By Algorithm 2.2, symbols a_{i+1} and a_i are on the same row of $P_{\text{hypo}}(w)$ (for any $w \in B^*$) if and only if the word w does not contain a symbol a_i somewhere to the right of a symbol a_{i+1} (since inserting this symbol a_i results in the part of the quasi-ribbon tableau that contains a_{i+1} being glued *below* the entry a_i).

Suppose $uvuv$ contains a symbol a_i somewhere to the right of a symbol a_{i+1} . Then, regardless of whether these symbols both lie in u , both lie in v , or one lies in u and the other lies in v , the word $vuvu$ also contains a symbol a_i to the right of a symbol a_{i+1} . Similar reasoning establishes that if any of the words $uvuv$, $vuvu$, $uvvu$, or $uvvu$ contains a symbol a_i somewhere to the right of a symbol a_{i+1} , so do all the others. Hence either the symbols a_i and a_{i+1} are on the same row in all of $P_{\text{hypo}}(uvuv)$, $P_{\text{hypo}}(vuvu)$, $P_{\text{hypo}}(uvvu)$, and $P_{\text{hypo}}(uvvu)$, or on different rows in all of $P_{\text{hypo}}(uvuv)$, $P_{\text{hypo}}(vuvu)$, $P_{\text{hypo}}(uvvu)$, and $P_{\text{hypo}}(uvvu)$.

A quasi-ribbon tableau is clearly determined by its evaluation and by knowledge of whether adjacent different entries are on the same row. Thus, noting that $\text{ev}(uvuv) = \text{ev}(vuvu) = \text{ev}(uvvu) = \text{ev}(uvvu)$, it follows from the previous paragraph that

$$\begin{aligned} xyxy &= P_{\text{hypo}}(uvuv) = yxyx = P_{\text{hypo}}(vuvu) \\ &= yxyx = P_{\text{hypo}}(uvvu) = xyxy = P_{\text{hypo}}(uvvu). \quad \square \end{aligned}$$

3.2. Taiga monoid.

Proposition 3.2. *Let $p, q, r \in \text{taig}$ be such that $\text{ev}(p) = \text{ev}(q) = \text{ev}(r)$. Then $pr = qr$.*

Proof. Let $u, v, w \in \mathcal{A}^*$ be words representing p, q, r , respectively. Since the tree $P_{\text{taig}}(x)$ is computed by applying Algorithm 2.5 to each symbol in x , proceeding right to left, it is clear that the order of the rightmost appearances of each symbol in x determines the shape of the tree $P_{\text{taig}}(x)$ and labels on the nodes. Since $\text{ev}(u) = \text{ev}(v) = \text{ev}(w)$, every symbol that appears in u or v also appears in w . Therefore $P_{\text{taig}}(uw)$ and $P_{\text{taig}}(vw)$ both have the same shape as $P_{\text{taig}}(w)$, and both have the same labels on each node. Since $\text{ev}(uw) = \text{ev}(vw)$, the multiplicities of symbols in $P_{\text{taig}}(uw)$ and in $P_{\text{taig}}(vw)$ are equal. Thus $P_{\text{taig}}(uw)$ and $P_{\text{taig}}(vw)$ are equal. Hence $pr = P_{\text{stal}}(uw) = P_{\text{stal}}(vw) = qr$. \square

Corollary 3.3. *The taiga monoid satisfies the non-trivial identity $xyxy = yxyx$.*

Proof. Let $x, y \in \text{taig}$. Let $p = r = xy$ and $q = yx$. Then $\text{ev}(p) = \text{ev}(q) = \text{ev}(r)$. Thus, by Proposition 3.2, $xyxy = pr = qr = yxyx$. \square

3.3. Sylvester monoid.

Proposition 3.4. *Let $p, q, r \in \text{sylv}$ be such that $\text{ev}(p) = \text{ev}(q) = \text{ev}(r)$. Then $pr = qr$.*

Proof. Let $u, v, w \in \mathcal{A}^*$ be words representing p, q, r , respectively. The tree $P_{\text{sylv}}(x)$ is computed by applying Algorithm 2.3 to each symbol in x , proceeding right to left. That is, $P_{\text{sylv}}(uw)$ and $P_{\text{sylv}}(vw)$ are obtained by inserting

u and v (respectively) into $P_{\text{sylv}}(w)$. Since $\text{ev}(u) = \text{ev}(v) = \text{ev}(w)$, every symbol that appears in u or v also appears in w and thus in $P_{\text{sylv}}(w)$.

Consider how further symbols from u or v are inserted into $P_{\text{sylv}}(w)$:

- Let a be the smallest symbol that appears in u , v , and w . Then the leftmost node of $P_{\text{sylv}}(w)$ must be labelled by a . Thus, during the computation of $P_{\text{sylv}}(uw)$ and $P_{\text{sylv}}(vw)$, all symbols a in u and v will certainly be inserted into the left subtree of this leftmost node in $P_{\text{sylv}}(w)$.
- Let c be some symbol other than a that appears in u , v , and w , and let b be the maximum symbol less than c appearing in u , v , and w . Let N_c be the leftmost node in $P_{\text{sylv}}(w)$ labelled by c and let N_b be the rightmost node in $P_{\text{sylv}}(w)$ labelled by b . By the maximality of b , there is no node in $P_{\text{sylv}}(w)$ that is to the right of N_b and to the left of N_c . Thus the lowest node in $P_{\text{sylv}}(w)$ that is above or equal to both N_b and N_c must be one of N_b or N_c . That is, one of the following must hold:
 - N_b is above N_c . Then N_c is in the right subtree of N_b , since $b < c$. Since there is no node that is to the right of N_b and to the left of N_c , it follows that N_c has an empty left subtree. Thus any symbol c in u or v will be inserted into this currently empty left subtree of N_c .
 - N_c is above N_b . Then N_b is in the left subtree of N_c , since $b < c$. Since there is no node that is to the right of N_b and to the left of N_c , it follows that N_b is the left child of N_c , and that N_b has an empty right subtree. Thus any symbol c in u or v will be inserted into this currently empty right subtree of N_b .

Combining these cases, one sees that every symbol d from u or v is inserted into a particular previously empty subtree of $P_{\text{sylv}}(w)$, dependent only on the value of the symbol d (and not on its position in u or v), and that unequal symbols are inserted into different subtrees. Since $\text{ev}(u) = \text{ev}(v)$, the same number of symbols d are inserted, for each such symbol d . Hence $P_{\text{sylv}}(uw) = P_{\text{sylv}}(vw)$ and thus $pr = P_{\text{stal}}(uw) = P_{\text{stal}}(vw) = qr$. \square

Following the reasoning in the proof of Corollary 3.3, but using Proposition 3.4 instead of Proposition 3.2, one obtains the following result:

Corollary 3.5. *The sylvester monoid satisfies the non-trivial identity $xyxy = yxxy$.*

3.4. Stalactic monoid.

Proposition 3.6. *Let $p, q, r \in \text{stal}$ be such that $\text{ev}(p) = \text{ev}(q) = \text{ev}(r)$. Then $pr = qr$.*

Proof. Let $u, v, w \in \mathcal{A}^*$ be words representing p, q, r , respectively. From Algorithm 2.4, it is clear that it is the rightmost appearance of each symbol that determines the order of symbols in the first row of a stalactic tableau. Since $\text{ev}(u) = \text{ev}(v) = \text{ev}(w)$, every symbol that appears in u or v also appears in w . It therefore follows that $P_{\text{stal}}(uw)$ and $P_{\text{stal}}(vw)$ have identical

first rows. Since $\text{ev}(uw) = \text{ev}(vw)$, the number of appearances of each symbol in $\text{P}_{\text{stal}}(uw)$ and $\text{P}_{\text{stal}}(vw)$ are equal. Hence $pr = \text{P}_{\text{stal}}(uw) = \text{P}_{\text{stal}}(vw) = qr$. \square

Again following the reasoning in the proof of Corollary 3.3, but using Proposition 3.6 instead of Proposition 3.2, one obtains the following result:

Corollary 3.7. *The stalactic monoid satisfies the non-trivial identity $xyxy = yxxy$.*

3.5. Baxter monoid.

Proposition 3.8. *Let $p, q, r, s \in \text{baxt}$ be such that $\text{ev}(p) = \text{ev}(q) = \text{ev}(r) = \text{ev}(s)$. Then $spr = sqr$.*

Proof. Suppose $spr = (T_L^{(1)}, T_R^{(1)})$ and $sqr = (T_L^{(2)}, T_R^{(2)})$. Let $p' = sp$ and $q' = sq$; note that $\text{ev}(p') = \text{ev}(q')$. By Proposition 3.4 applied to the elements p', q', r , it follows that $T_R^{(1)} = \text{P}_{\text{sylv}}(p'r) = \text{P}_{\text{sylv}}(q'r) = T_R^{(2)}$. Using reasoning symmetrical to the proof of Proposition 3.4 for insertion into a left strict binary search tree, and considering the elements $p'' = pr$, $q'' = qr$, and s , it follows that $T_L^{(1)} = T_L^{(2)}$. Hence $spr = sqr$. \square

Corollary 3.9. *The Baxter monoid satisfies the identity $yxxyxy = yxyxxy$.*

Proof. Let $x, y \in \text{baxt}$. Let $p = r = xy$ and $q = s = yx$. Then $\text{ev}(p) = \text{ev}(q) = \text{ev}(r) = \text{ev}(s)$. Thus, by Proposition 3.8, $yxxyxy = spr = sqr = yxyxxy$. \square

3.6. Bell monoid. The present authors and Silva [CMS] have shown that the elements $\text{P}_{\text{bell}}(21)$ and $\text{P}_{\text{bell}}(1)$ generate a free submonoid of bell . Since free monoids of rank at least 2 satisfy no non-trivial identities, it follows that bell also satisfies no non-trivial identities. Let $\text{bell}_n = \mathcal{A}_n^* / \equiv_{\text{bell}}$ (where \equiv_{bell} is naturally restricted to $\mathcal{A}_n^* \times \mathcal{A}_n^*$) be the *Bell monoid of rank n* ; clearly bell_n is a submonoid of bell . Then in fact bell_n for $n \geq 2$ satisfies no non-trivial identities, while bell_1 , as a monogenic monoid, is of course commutative and satisfies the identity $xy = yx$.

3.7. Plactic monoid. Whether the plactic monoid satisfies a non-trivial identity remains an open question, but some partial results are known. Let $\text{plac}_n = \mathcal{A}_n^* / \equiv_{\text{plac}}$ (where \equiv_{plac} is naturally restricted to $\mathcal{A}_n^* \times \mathcal{A}_n^*$) be the *plactic monoid of rank n* ; clearly plac_n is a submonoid of plac .

First, plac_1 is monogenic and thus commutative and satisfies $xy = yx$. Kubat & Okniński [KO14] have shown that plac_2 satisfies Adian's identity $xyxyxyxyx = xyxyxyxyx$, and that plac_3 satisfies the identity $pqqppq = pqpqqp$, where $p(x, y)$ and $q(x, y)$ are respectively the left and right side of Adian's identity (and so the identity $pqqppq = pqpqqp$ has sixty variables x or y on each side). Furthermore, plac_3 does not satisfy Adian's identity [KO14, p. 111–2].

Question 3.10. For each $n \geq 4$, is there a non-trivial identity satisfied by plac_n ?

Question 3.11. Is there a non-trivial identity satisfied by plac ?

REFERENCES

- [Adi66] S. Adian. ‘Defining relations and algorithmic problems for groups and semigroups’. *Trudy Mat. Inst. Steklov*, 85 (1966), pp. 3–123. URL: <http://www.mathnet.ru/eng/tm2766>.
- [CEK⁺01] J. Cassaigne, M. Espie, D. Krob, J.-C. Novelli, & F. Hivert. ‘The Chinese Monoid’. *Int. J. Alg. Comput.*, 11, no. 03 (2001), pp. 301–334. DOI: 10.1142/S0218196701000425.
- [CM] A. J. Cain & A. Malheiro. ‘Crystallizing the hypoplactic monoid: from quasi-Kashiwara operators to the Robinson–Schensted–Knuth-type correspondence for quasi-ribbon tableaux’. *Journal of Algebraic Combinatorics*. DOI: 10.1007/s10801-016-0714-6.
- [CMS] A. J. Cain, A. Malheiro, & F. Silva. ‘On the bell monoid’. In preparation.
- [DK94] G. Duchamp & D. Krob. ‘Plactic-growth-like monoids’. In M. Ito & H. Jürgensen, eds, *Words, languages and combinatorics, II*, p. 124–142, River Edge, NJ, 1994. World Scientific.
- [Ful97] W. Fulton. *Young Tableaux: With Applications to Representation Theory and Geometry*. No. 35 in *LMS Student Texts*. Cambridge University Press, 1997.
- [Gir12] S. Giraudo. ‘Algebraic and combinatorial structures on pairs of twin binary trees’. *J. Algebra*, 360 (2012), pp. 115–157. DOI: 10.1016/j.jalgebra.2012.03.020.
- [HNT05] F. Hivert, J.-C. Novelli, & J.-Y. Thibon. ‘The algebra of binary search trees’. *Theoret. Comput. Sci.*, 339, no. 1 (2005), pp. 129–165. DOI: 10.1016/j.tcs.2005.01.012.
- [HNT07] F. Hivert, J.-C. Novelli, & J.-Y. Thibon. ‘Commutative combinatorial Hopf algebras’. *J. Algebr. Comb.*, 28, no. 1 (2007), pp. 65–95. DOI: 10.1007/s10801-007-0077-0.
- [Jed11] F. Jedrzejewski. ‘Plactic classification of modes’. In C. Agon, M. Andreatta, G. Assayag, E. Amiot, J. Bresson, & J. Mandereau, eds, *Mathematics and Computation in Music*, Lecture Notes in Comput. Sci., pp. 350–353. Springer, 2011. DOI: 10.1007/978-3-642-21590-2_31.
- [JO11] J. Jaszńska & J. Okniński. ‘Structure of Chinese algebras’. *J. Algebra*, 346, no. 1 (2011), pp. 31–81. DOI: 10.1016/j.jalgebra.2011.08.020.
- [KO14] Ł. Kubat & J. Okniński. ‘Identities of the plactic monoid’. *Semigroup Forum*, 90, no. 1 (2014), pp. 100–112. DOI: 10.1007/s00233-014-9609-9.
- [Lot02] M. Lothaire. *Algebraic Combinatorics on Words*. No. 90 in *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2002.
- [LS78] A. Lascoux & M.-P. Schützenberger. ‘Sur une conjecture de H. O. Foulkes’. *C. R. Acad. Sci. Paris Sér. A-B*, 286, no. 7 (1978), pp. A323–A324.
- [LS81] A. Lascoux & M.-P. Schützenberger. ‘Le monoïde plaxique’. In *Noncommutative structures in algebra and geometric combinatorics*, Quaderni de “La Ricerca Scientifica”, pp. 129–156, Rome, 1981. CNR. URL: <http://igm.univ-mlv.fr/~berstel/Mps/Travaux/A/1981-1PlaxiqueNaples.pdf>.
- [LS85] A. Lascoux & M.-P. Schützenberger. ‘Schubert polynomials and the Littlewood–Richardson rule’. *Lett. Math. Phys.*, 10, no. 2-3 (1985), pp. 111–124. DOI: 10.1007/BF00398147.
- [LS90] A. Lascoux & M.-P. Schützenberger. ‘Tableaux and noncommutative Schubert polynomials’. *Funct. Anal. Its Appl.*, 23, no. 3 (1990), pp. 223–225. DOI: 10.1007/BF01079531.
- [Mac08] I. Macdonald. *Symmetric Functions and Hall Polynomials*. Clarendon Press, Oxford University Press, 2008.
- [Nov00] J.-C. Novelli. ‘On the hypoplactic monoid’. *Discrete Mathematics*, 217, no. 1-3 (2000), pp. 315–336. DOI: 10.1016/S0012-365X(99)00270-8.
- [Pri13] J.-B. Prieze. ‘A lattice of combinatorial Hopf algebras: Binary trees with multiplicities’. In *Formal Power Series and Algebraic Combinatorics*, Nancy, 2013. The Association. Discrete Mathematics & Theoretical Computer Science. URL: <http://www.dmtcs.org/pdftpapers/dmAS0196.pdf>.

- [Rey07] M. Rey. ‘Algebraic constructions on set partitions’. In *Formal Power Series and Algebraic Combinatorics*, 2007. URL: <http://www-igm.univ-mlv.fr/~rey/articles/rey-fpsac07.pdf>.

CENTRO DE MATEMÁTICA E APLICAÇÕES, FACULDADE DE CIÊNCIAS E TECNOLOGIA,
UNIVERSIDADE NOVA DE LISBOA, 2829-516 CAPARICA, PORTUGAL

E-mail address: `a.cain@fct.unl.pt`

DEPARTAMENTO DE MATEMÁTICA & CENTRO DE MATEMÁTICA E APLICAÇÕES, FAC-
ULDADE DE CIÊNCIAS E TECNOLOGIA, UNIVERSIDADE NOVA DE LISBOA, 2829-516 CA-
PARICA, PORTUGAL

E-mail address: `ajm@fct.unl.pt`